Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

Avvertenze per la consegna: apporre all'inizio di ogni file sorgente un commento contenente i propri dati (**cognome, nome, numero di matricola**) e il **numero** della prova d'esame. Al termine, **consegnare tutti i file sorgenti** necessari alla compilazione e alla corretta esecuzione del programma.
Nota: **il main non è opzionale**; i *test* richiesti vanno implementati.
Consiglio: per verificare l'assenza di *warning*, eseguire sempre *"Rebuild All"*.

L'azienda Pego ha inventato un sofisticato sistema di mattoncini ad incastro, con cui è possibile costruire case, oggetti, auto, astronavi, etc. Per facilitare le vendite, la Pego ha creato delle scatole contenenti più mattoncini, con istruzioni, per costruire uno specifico oggetto. Tali scatole sono dette "set", e sono identificate con un numero intero. Alcuni set sono detti "semplici": contengono direttamente mattoncini e istruzioni. Altri set sono detti "complessi": infatti sono in realtà come insiemi di altri set semplici.
Un negozio di giocattoli è specializzato nella vendita di set Pego, e quindi tiene traccia in un file di testo di quali set tiene in negozio. In particolare, su ogni riga, scrive le seguenti informazioni: **l'identificativo** unico del set Pego (un intero); separato da uno spazio, una descrizione sommaria del **contenuto** (una stringa di al più 2047 caratteri utili, senza spazi); a seguire, separato da uno spazio, un carattere 'S' oppure 'C', che indicano il **tipo** di set (semplice o complesso, un char); ancora, separato da uno spazio, il numero di scatole **disponibili** in negozio (un intero); infine, sempre separato da uno spazio, il **costo** del set (un float).
In un secondo file di testo viene specificato come sono composte le scatole complesse. Su ogni riga è specificato per primo l'identificativo di una scatola complessa, e a seguire, tutti separati da uno spazio, gli identificativi dei set semplici che compongono tale set complesso. Ogni set complesso può essere composto al massimo da dieci set semplici. Ogni riga, per comodità, è sempre terminata dal numero intero -1000, il cui significato è che la composizione del set complesso corrente è terminata.
Si vedano, a titolo di esempio, i file "`negozio.txt`" e "`complessi.txt`" forniti con lo StartKit.

Esercizio 1 – Struttura dati Set, e funzioni di lett./scritt. (mod. element.h e negozio.h/.c)

Si definisca una struttura dati `Set` per memorizzare i dati relativi ad una scatola Pego, cioè l'id della scatola, la stringa che ne descrive il contenuto, il tipo di set, le scatole disponibili, e il costo.
Si definisca poi una struttura dati `Complex`, destinata a memorizzare la composizione di un Set complesso: la struttura dovrà quindi memorizzare l'id del set complesso, un array di dimensione fisica pari a 10 per indicare i set semplici che lo compongono, e un intero ad indicare la dimensione logica dell'array. Nell'array verranno memorizzati i soli id dei set semplici che compongono il `Complex`.
Si definisca la funzione:

```
Set leggiUnSet(FILE * fp);
```

che, ricevuto in ingresso un puntatore ad una struttura dati di tipo `FILE`, contenente i set in negozio, legga i dati contenuti su una sola riga e li restituisca tramite una struttura dati di tipo `Set`. Qualora vi siano problemi nella lettura, la funzione restituisca una struttura dati con identificativo del `Set` pari a -1000. Nella struttura dati restituita come risultato, il candidato abbia cura anche di rendere maggiormente "human readable" il campo di descrizione del contenuto: in particolare, si abbia cura di sostituire i caratteri '_' (underscore) con uno spazio.

Si definisca la funzione:

```
list leggiTuttiSet(char * fileName);
```

che, ricevuto in ingresso il nome di un file che contenga l'elenco dei set in negozio, legga da tale file i dati di tutti i set e li restituisca tramite una lista di strutture dati di tipo `Set`. Qualora la funzione incontri dei problemi nella lettura, o vi siano errori nell'apertura del file, la funzione dovrà stampare un messaggio di errore a video, e restituire una lista vuota.

Il candidato abbia cura di realizzare nel main opportuni test al fine di verificare il corretto funzionamento delle funzioni di cui sopra, avendo cura di deallocare la memoria, se necessario.

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

Esercizio 2 –Composizione dei Set complessi (moduli element.h/.c e negozio.h/.c)

Si definisca la funzione:

```
Complex leggiUnComplex(FILE * fp);
```

che, ricevuto in ingresso un puntatore ad una struttura dati di tipo **FILE**, legga i dati contenuti su una sola riga e li restituisca tramite una struttura dati di tipo **Complex**. Qualora vi siano problemi nella lettura, la funzione restituisca una struttura dati con identificativo del **Complex** pari a -1000.

Il candidato definisca una funzione:

```
Complex trovaComplex(char * fileName, int target);
```

che, ricevuti in ingresso il nome di un file, e l'identificativo di un set complesso (parametro **target**), restituisca in uscita una struttura dati di tipo **Complex**, contenente la composizione del set specificato come **target**. Ovviamente, il file specificato tramite il parametro **fileName** conterrà l'elenco dei **Set** complessi con la loro composizione, come specificato nell'introduzione.

La funzione abbia cura di far si che l'array che descrive la composizione del **target** sia ordinato in ordine decrescente. A tal scopo, si usi l'algoritmo "quick sort" visto a lezione.

Qualora il **target** non sia presente nel file, la funzione restituisca una struttura con identificatore pari al valore -1000. Qualora vi siano problemi nell'accedere al file, la funzione restituisca sempre una struttura con identificatore pari a -1000.

Esercizio 3 – Verifica disponibilità (modulo negozio.h/negozio.c)

Si sviluppi una funzione:

```
int disp(list elenco, int target, char * fileName);
```

che, ricevuti in ingresso la lista **elenco** dei **Set** in negozio, e l'identificatore di un set **target** desiderato, determini se sia possibile acquistare tale **Set** oppure no. Tramite il parametro **fileName**, alla funzione viene specificato il nome del file contenente la composizione dei **Set** complessi (si veda dopo).

La funzione dovrà gestire quattro possibili situazioni:

a) Il set **target** non è presente nella lista **elenco**: la funzione dovrà restituire un valore interpretabile come "falso", poiché il set non è acquistabile.
b) Il set **target** è presente nella lista **elenco**, con disponibilità maggiore di zero: la funzione dovrà restituire un valore interpretabile come "vero".
c) Il set **target** è presente nella lista **elenco**, con disponibilità pari a zero, ed il set è un set semplice: la funzione dovrà restituire un valore "falso".
d) Il set **target** è presente nella lista **elenco**, con disponibilità pari a zero, ma il set è un set complesso: in questo caso, il negozio potrebbe "comporre" il set complesso mettendo insieme i set semplici; a tal fine, bisogna però verificare che tali set semplici siano a loro volta disponibili.

Si suggerisce di realizzare una funzione:

```
int check(list elenco, int target, Set * theSet);
```

che, ricevuto l'identificatore **target** di un set, restituisca un valore interpretabile come vero se il set è presente in **elenco** (indipendentemente dalla disponibilità). Tramite il parametro **theSet** passato per riferimento, la funzione restituisca la struttura dati **Set** relativa al **target** specificato.

Esercizio 4 Stampa dei risultati, e de-allocazione memoria (main.c)

Il candidato realizzi nella funzione **main(…)** un programma che stampi a video la lista dei **Set** memorizzati nel file "negozio.txt". Il programma poi chieda all'utente di specificare un **Set target**, e stampi a video un messaggio che dica all'utente se è possibile acquistare il set o no (si utilizzi quindi la funzione di cui al l'esercizio 3).

Al termine del programma, il candidato abbia cura di de-allocare tutta la memoria allocata dinamicamente, ivi compresa la memoria allocata per le liste.

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

"element.h":

```
#define _CRT_SECURE_NO_WARNINGS

#ifndef _ELEMENT_H
#define _ELEMENT_H

#define DIM 2048

typedef struct {
	int id;
	char contenuto[DIM];
	char tipo;
	int disp;
	float costo;
} Set;

typedef Set element;

typedef struct {
	int id;
	int subset[10];
	int dimLogica;
} Complex;

int compare(int id1, int id2);

#endif
```

"element.c":

```
#include "element.h"


int compare(int id1, int id2) {
	return id2 - id1;
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

"list.h"

```
#ifndef LIST_H
#define LIST_H

#include "element.h"

typedef struct list_element {
    element value;
    struct list_element  *next;
} item;

typedef  item* list;

typedef  int boolean;

/* PRIMITIVE  */
list emptylist(void);
boolean empty(list);
list cons(element, list);
element head(list);
list tail(list);

void showlist(list l);
void freelist(list l);
int member(element el, list l);

list insord_p(element el, list l);

#endif
```

"list.c":

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "list.h"

/* OPERAZIONI PRIMITIVE */
list  emptylist(void)            /* costruttore lista vuota */
{
    return NULL;
}

boolean  empty(list l)     /* verifica se lista vuota */
{
    return (l==NULL);
}

list  cons(element e, list l)
{
  list t;       /* costruttore che aggiunge in testa alla lista */
  t=(list)malloc(sizeof(item));
  t->value=e;
  t->next=l;
  return(t);
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

```
element head(list l) /* selettore testa lista */
{
  if (empty(l)) exit(-2);
  else return (l->value);
}

list  tail(list l)         /* selettore coda lista */
{
  if (empty(l)) exit(-1);
  else return (l->next);
}

void showlist(list l) {
       element temp;
       if (!empty(l)) {
              temp = head(l);
              printf("%d: %s, Tipo:%c, Disp:%d, Costo:%.2f\n",
                     temp.id, temp.contenuto, temp.tipo, temp.disp, temp.costo);
              showlist(tail(l));
              return;
       }
       else {
              printf("\n\n");
              return;
       }
}

int member(element el, list l) {
       int result = 0;
       while (!empty(l) && !result) {
              result = (el.id == head(l).id);
              if (!result)
                     l = tail(l);
       }
       return result;
}

void freelist(list l) {
       if (empty(l))
              return;
       else {
              freelist(tail(l));
              free(l);
       }
       return;
}

list insord_p(element el, list l) {
       list pprec = NULL, patt = l, paux;
       int trovato = 0;

       while (patt!=NULL && !trovato) {
              if (el.costo > patt->value.costo)
                     trovato = 1;
              else {
                     pprec = patt;
```

```
                patt = patt->next;
            }
        }
        paux = (list) malloc(sizeof(item));
        paux->value = el;
        paux->next = patt;
        if (patt==l)
            return paux;
        else {
            pprec->next = paux;
            return l;
        }
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

"negozio.h":

```
#define _CRT_SECURE_NO_WARNINGS
#pragma once
#ifndef _NEGOZIO_H
#define _NEGOZIO_H


#include "element.h"
#include "list.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Es. 1
Set leggiUnSet(FILE * fp);
list leggiTuttiSet(char * fileName);

// Es. 2
Complex leggiUnComplex(FILE * fp);
Complex trovaComplex(char * fileName, int target);
void quickSort(int a[], int dim);
void quickSortR(int a[], int iniz, int fine);

// Es. 3
int check(list elenco, int target, Set * theSet);
int disp(list elenco, int target, char * fileName);

#endif
```

"negozio.c":

```
#include "negozio.h"

Set leggiUnSet(FILE * fp) {
	Set result;
	int ok;
	int i;

	ok = fscanf(fp, "%d%s %c %d%f", &(result.id), result.contenuto, &(result.tipo),
&(result.disp), &(result.costo));
	if (ok != 5) {
		result.id = -1000;
	}
	else {
		for (i = 0; i < strlen(result.contenuto); i++) {
			if (result.contenuto[i] == '_')
				result.contenuto[i] = ' ';
		}
	}
	return result;
}

list leggiTuttiSet(char * fileName) {
	FILE * fp;
```

```
        list result;
        Set temp;

        result = emptylist();

        fp = fopen(fileName, "rt");
        if (fp != NULL) {
                temp = leggiUnSet(fp);
                while (temp.id >= 0) {
                        result = cons(temp, result);
                        temp = leggiUnSet(fp);
                }
                fclose(fp);
        }
        else {
                printf("Problema nell'apertura del file %s\n", fileName);
        }
        return result;
}


// Es. 2
Complex leggiUnComplex(FILE * fp) {
        Complex result;
        int temp;

        if (fscanf(fp, "%d", &(result.id)) == 1) {
                result.dimLogica = 0;
                while ((fscanf(fp, "%d", &temp) == 1) && (temp >= 0)) {
                        result.subset[result.dimLogica] = temp;
                        (result.dimLogica)++;
                }
        }
        else {
                result.id = -1000;
        }
        return result;
}

Complex trovaComplex(char * fileName, int target) {
        FILE * fp;
        Complex temp;

        fp = fopen(fileName, "rt");
        if (fp != NULL) {
                temp = leggiUnComplex(fp);
                while (temp.id != -1000 && temp.id != target) {
                        temp = leggiUnComplex(fp);
                }
                if (temp.id == target) {
                        quickSort(temp.subset, temp.dimLogica);
                }
                fclose(fp);
        }
        else {
                printf("Prblema nell'apertura del file %s\n", fileName);
                temp.id = -1000;
```

```
	}
	return temp;
}

void scambia(int *a, int *b) {
	int tmp = *a;
	*a = *b;
	*b = tmp;
}

void quickSort(int a[], int dim) {
	quickSortR(a, 0, dim - 1);
}

void quickSortR(int a[], int iniz, int fine) {
	int i, j, iPivot, pivot;
	if (iniz < fine) {
		i = iniz;
		j = fine;
		iPivot = fine;
		pivot = a[iPivot];
		do {
			while (i < j && compare(a[i], pivot)<=0) i++;
			while (j > i && compare(a[j], pivot)>=0) j--;
			if (i < j) scambia(&a[i], &a[j]);
		} while (i < j);
		if (i != iPivot && a[i] != a[iPivot]) {
			scambia(&a[i], &a[iPivot]);
			iPivot = i;
		}
		if (iniz < iPivot - 1)
			quickSortR(a, iniz, iPivot - 1);
		if (iPivot + 1 < fine)
			quickSortR(a, iPivot + 1, fine);
	}
}

// Es. 3
int check(list elenco, int target, Set * theSet) {
	int trovato = 0;

	while (!empty(elenco) && !trovato) {
		if (head(elenco).id == target)
			trovato = 1;
		else
			elenco = tail(elenco);
	}
	if (trovato)
		*theSet = head(elenco);
	return trovato;
}

int disp(list elenco, int target, char * fileName) {
	Set tempSet;
	int trovato;
	int result;
	Complex complex;
```

```
        int i;

        trovato = check(elenco, target, &tempSet);
        if (!trovato) { // caso (a)
                result = 0;
        }
        else { // casi (b), (c), e (d)
                if (tempSet.disp > 0) // caso (b)
                        result = 1;
                else { // casi (c) e (d)
                        if (tempSet.tipo == 'S')  // caso (c)
                                result = 0;
                        else { // caso (d)
                                complex = trovaComplex(fileName, target);
                                for (i = 0; i < complex.dimLogica && trovato; i++) {
                                        trovato = check(elenco, complex.subset[i], &tempSet);
                                }
                                result = trovato;
                        }
                }
        }
        return trovato;
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

"main.c":

```
#define _CRT_SECURE_NO_WARNINGS

#include "element.h"
#include "list.h"
#include "negozio.h"

int main() {
	{ // Es. 1
		list elenco;
		elenco = leggiTuttiSet("negozio.txt");
		showlist(elenco);
		freelist(elenco);
	}
	{ // Es. 2
		Complex temp;
		int i;
		temp = trovaComplex("complessi.txt", 67914);
		printf("\n\n");
		if (temp.id >= 0) {
			printf("Set complesso %d:\n", 67914);
			for (i = 0; i < temp.dimLogica; i++) {
				printf("\t%d\n", temp.subset[i]);
			}
		}
		else {
			printf("Non ho trovato il set %d\n", 67914);
		}
		printf("\n\n");
	}
	{ // Es. 3
		list elenco;
		int target;
		int trovato;

		elenco = leggiTuttiSet("negozio.txt");

		printf("A quale Set si e' interessati? ");
		scanf("%d", &target);
		trovato = disp(elenco, target, "complessi.txt");
		if (!trovato) {
			printf("Spiacenti, il Set %d non e' disponibile\n", target);
		}
		else {
			printf("Il Set %d e' disponiible\n", target);
		}
		freelist(elenco);
	}
	return 0;
}
```

Fondamenti di Informatica T-1, 2019/2020 – Modulo 2

Prova d'Esame 1A di Giovedì 9 Gennaio 2020 – tempo a disposizione 2h

"negozio.txt":

```
95645 astronave_con_jedi_star_wars_IX C 2 499.00
95000 astronave_star_wars_IX S 1 459.00
645 jedi_star_wars_IX S 1 69.00
67000 medioevo_castello S 1 199.00
67914 medioevo_castello_con_drago C 0 220.00
914 medioevo_drago S 1 35.00
```

"complessi.txt":

```
95645 95000 600 45 -1000
48960 48000 900 60 -1000
67914 914 67000 -1000
```